Introduction aux systèmes de type UNIX

Bastien Boussau*& Leonor Palmeira†

Résumé

Le but de ce TP est d'acquérir les connaissances de base nécessaires pour comprendre et utiliser une machine UNIX, mais aussi de vous rendre autonome dans votre apprentissage de ces systèmes, et de vous apprendre à chercher les réponses à vos questions.

Table des matières

1 Petit historique **3**
- 1.1 Qu'est-ce qu'un système d'exploitation ? . . . 3
  - 1.1.1 Qu'est-ce que Windows ? . . . 3
  - 1.1.2 Qu'est-ce que Mac OS ? . . . 3
  - 1.1.3 Qu'est-ce que Linux ? . . . 4
  - 1.1.4 Mais alors qu'est-ce qu'UNIX ? . . . 4

2 Kit de survie UNIX **5**
- 2.1 Save Our Souls – S.O.S. . . . 5
- 2.2 Arborescence . . . 5
  - 2.2.1 Se déplacer dans un autre dossier . . . 5
  - 2.2.2 Lister l'ensemble des fichiers d'un dossier . . . 6
- 2.3 Gain de temps . . . 6
  - 2.3.1 Complétion . . . 6
  - 2.3.2 Historique . . . 6
- 2.4 Gestion des dossiers et des fichiers . . . 7
  - 2.4.1 Gestion des dossiers . . . 7
  - 2.4.2 Gestion des fichiers . . . 7
- 2.5 Droits . . . 8
  - 2.5.1 Visualiser les permissions . . . 9
  - 2.5.2 Modifier les permissions . . . 9
- 2.6 Gestion des processi . . . 9
  - 2.6.1 Lister les processi . . . 9
  - 2.6.2 Arrière et premier plans . . . 10
  - 2.6.3 Procicide . . . 10

*boussau@biomserv.univ-lyon1.fr
†palmeira@biomserv.univ-lyon1.fr

3 Approfondissement UNIX **10**
3.1 Manipuler des fichiers texte 10
3.1.1 Visualisation de fichiers texte 11
3.1.2 Concaténation de fichiers 11
3.2 Automatiser des tâches 11
3.2.1 Les langages *shell* 11
3.2.2 Petits exercices 12
3.2.3 Alias 13

1 Petit historique

1.1 Qu'est-ce qu'un système d'exploitation ?

Un ordinateur peut être considéré comme une machine de calcul, constituée de nombreux composants agissant ensemble de manière coordonnée. Le système d'exploitation (*operating system*) est la couche logicielle permettant de coordonner l'utilisation de ce matériel informatique (*hardware*). Il va, non seulement gérer les processi et leur existence simultanée en optimisant l'utilisation des éléments du système, mais aussi offrir à l'utilisateur une interface conviviale.

> Un **processus** est défini, de manière schématique, par un ensemble d'instructions à exécuter (un programme) et un espace mémoire pour l'exécution de ces instructions. De manière plus pratique, il s'agit d'une tâche en exécution, ou en cours d'exécution.

Il est généralement constitué d'un noyau (*kernel*), qui interagit directement avec le matériel, et qui est la première couche logicielle à être chargée lors de la mise sous tension de la machine, et d'une deuxième couche logicielle permettant l'interaction avec l'utilisateur.

1.1.1 Qu'est-ce que Windows ?

Windows est une gamme de sytèmes d'exploitation développée par l'entreprise Microsoft. Les premières versions étaient constituées d'un noyau MS-DOS, et d'une sur-couche logicielle. Le noyau MS-DOS a depuis été abandonné, et les systèmes actuels semblent être inspirés d'autres noyaux existants.

1.1.2 Qu'est-ce que Mac OS ?

Mac OS (*Macintosh Operating System*) est une gamme de systèmes d'exploitation graphiques développés par l'entreprise Apple Computer pour leurs ordinateurs. L'interface graphique, et sa maniabilité, a toujours été

l'un des grands atouts mis en avant par la société. Actuellement, la version Mac OS X est construite sur un noyau inspiré du noyau UNIX.

1.1.3 Qu'est-ce que Linux ?

Les systèmes d'exploitation dits Linux devraient, en fait, être nommés GNU/Linux, car il s'agit généralement d'un système d'exploitation quasi-complet lié au projet GNU, associé à un noyau Linux.

Le projet GNU (acronyme pour *GNU's Not UNIX*), lancé en 1985 par Richard Stallman (photo de gauche) visait au développement d'un système d'exploitation complet et libre de logiciels compatible UNIX, et comprenait donc le développement d'un noyau propre au projet. Mais le développement de ce dernier s'est avéré beaucoup plus long et difficile que prévu, et, au début des années 1990, de nombreux utilisateurs ont commencé à adopter le noyau récemment développé par Linus Torvalds (photo de droite). L'arrivée du noyau Linux a ainsi permis de mettre à disposition du grand public l'ensemble des logiciels issus du projet GNU.

1.1.4 Mais alors qu'est-ce qu'UNIX ?

C'est le nom d'un système d'exploitation créé en 1969 à but essentiellement professionnel dans les laboratoires AT&T Bell. Il devait être essentiellement : **portable** (utilisable sur de très nombreuses machines différentes, sans problèmes de compatibilité), **multi-tâches** (gestion de plusieurs processus en quasi-simultané) et **multi-utilisateurs** (utilisation simultané de l'ordinateur par plusieurs utlisateurs). Il a donné naissance à un standard, à un langage de programmation (le C), et de très nombreux systèmes sont nés en s'inspirant de sa philospohie.

L'ensemble des systèmes de type UNIX possèdent des caractéristiques qui les distinguent d'autres systèmes : l'interpréteur de commandes est l'outil de base du système ; de nombreux fichiers sont de simples fichiers texte ; le système de fichiers est hiérarchique...

2 Kit de survie UNIX

Lorsque l'on se retrouve confronté à l'utilisation d'une machine de type UNIX en ligne de commandes, l'apparence de ce type d'interface apparaît de prime abord bien moins sympathique que les interfaces dans lesquelles la navigation se fait à la souris. Mais lorsqu'on commence à vouloir traiter des données en grande quantité – comme c'est souvent le cas en bioinformatique – on s'aperçoit qu'il est plus agréable de taper quelques lignes dans une console pour automatiser une tâche plutôt que de traiter toutes les données à la main. Le but de cette partie est donc d'acquérir les connaissances de base nécessaires à l'utilisation de la ligne de commandes.

Pour commencer, il faut tout d'abord ouvrir une console qui permet d'entrer les commandes (demandez à l'encadrant si vous êtes perdus). Cette console (ou shell), est en fait un interpréteur de commandes, qui va traduire ce que vous écrivez en un ordre compréhensible par le système.

2.1 Save Our Souls – S.O.S.

Avant toute chose, apprenons à ouvrir un manuel d'aide. Toute commande (ou presque) dispose en effet d'une page de manuel, à laquelle on accède tout simplement en tapant : `man nom_commande`. Le manuel fournit des instructions très complètes sur le mode d'emploi de la commande. Pour faire défiler la page, on utilise les flèches du clavier, et pour quitter le manuel, on utilise la touche *q*.

Tapez dans une console `man pwd`.
Que fait la commande `pwd` ?
Exécutez-la.

2.2 Arborescence

Nous venons de voir la commande `pwd`. Maintenant, apprenons à naviguer dans l'arborescence des dossiers, à lister l'ensemble des fichiers d'un dossier, ... Tout ceci sans un seul clic de souris.

2.2.1 Se déplacer dans un autre dossier

La commande `cd` (change directory) permet de changer de dossier. Si on utilise `cd` sans argument, on est renvoyé dans son dossier "maison" (home), qui est le dossier dans lequel la console s'ouvre par défaut.

On peut ensuite naviguer dans l'arborescence, en spécifiant le chemin pour accéder au nouveau dossier, à partir du dossier dans lequel on se trouve. Le dossier courant est alors représenté par le symbole `"."`, et le dossier parent du dossier courant est représenté par `".."`.

Quel est votre dossier "maison" ?
Quel est son dossier parent ? S'y déplacer.

Vérifiez que vous êtes bien remonté d'un cran dans l'arborescence.

Revenez dans votre dossier "maison".

2.2.2 Lister l'ensemble des fichiers d'un dossier

La commande `ls` permet de faire la liste de l'ensemble des fichiers contenus dans un dossier. Si on l'utilise sans argument, la commande fait la liste des fichiers contenus dans le dossier courant. Si on l'utilise avec un dossier particulier en argument, la commande fait la liste de ses fichiers. De nombreuses options permettent de préciser les informations que fournit `ls` sur chacun des fichiers trouvés.

Listez l'ensemble des fichiers de votre dossier "maison".

`ls .`, `ls` et `ls -la` sont-ils équivalents ? Pourquoi ?

Listez l'ensemble des fichiers du dossier parent.

Représentez l'arborescence des dossiers depuis la racine du système de fichiers (trouvez la bonne option de la commande `ls`).

Revenez au dossier "maison".

2.3 Gain de temps

Deux mécanismes permettent de gagner un temps précieux lorsqu'on utilise la ligne de commandes : la complétion et l'historique.

2.3.1 Complétion

Lorsqu'il n'y a aucune ambiguïté dans la commande ou l'argument que vous souhaitez taper, le shell est capable de rechercher dans un ensemble de lieux définis les noms de fichiers qui peuvent compléter votre saisie. Cette complétion s'active au moyen de la touche *Tabulation*. Lorsqu'il y a ambiguïté, appuyer sur la touche *Tabulation* deux fois affiche un choix de commandes dont les noms complètent les premières lettres entrées.

Tapez dans une console

```
pw
```

puis tapez sur la touche *Tabulation* deux fois.

Combien existe-t-il de commandes dont les deux premières lettres sont `pw` ?

Retrouvez-vous la commande utilisée précédemment ?

Faites la même chose en tapant `ma` dans la console. Retrouvez-vous la commande `man` ?

2.3.2 Historique

Lorsqu'on est amené à taper de longues commandes dans la console, ou à relancer une commande que l'on vient d'effectuer, il est fort agréable de

pouvoir éviter de tout retaper. On peut pour ce faire utiliser l'historique qui récapitule l'ensemble des commandes qui ont été préalablement rentrées. On y accède en utilisant la touche *flèche montante*. De même, si on entre `history` dans une console, l'ensemble des commandes déjà entrées s'affiche à l'écran.

Affichez l'avant dernière commande que vous avez lancée, grâce à `history`. Inutile de la relancer.

Pensez maintenant à vous reposer les doigts en rappelant vos anciennes commandes avec la touche *flèche montante*.

2.4 Gestion des dossiers et des fichiers

2.4.1 Gestion des dossiers

- La commande `mkdir nomdossier` permet de créer le dossier "nomdossier".
- La commande `rmdir nomdossier` permet de supprimer le dossier "nomdossier".

Créez un dossier *Essais*. Placez-vous dans ce dossier pour la suite.

2.4.2 Gestion des fichiers

Nous allons d'abord créer un fichier. Pour ce faire, rien de plus simple que de "toucher" ce fichier. Dans la console, tapez `touch nomfichier`. Vous pouvez vérifier que ce fichier existe : `ls`.

Mieux encore, nous allons maintenant écrire dans ce fichier.

Dans la console, tapez :

`gedit nomfichier` **(on peut aussi essayer** `kedit`**), ou**

`emacs nomfichier` **(ou bien** `xemacs nomfichier`**), voire pour les aventuriers**

`vi nomfichier`**.**

Tapez quelques lignes dans le fichier et enregistrez-le.

> Que sont *gedit*, *kedit*, *emacs*, *xemacs*, ou *vi* ?
> Ce sont des **éditeurs de texte**, qui vous permettent de modifier vos fichiers. Il en existe des classiques, tels que : *gedit* ou *kedit*), qui ressemblent beaucoup au *bloc-notes* de Windows. Certains sont très puissants et très bien adaptés à la programmation, même si un peu moins intuitifs au début : *emacs* (ou *xemacs*). *vi* est lui aussi très puissant, mais tourne directement dans la console (pas de mode graphique), ce qui le rend fort peu intuitif, mais pratique lorsqu'on n'a pas accès à une interface graphique ou lorsque l'ordinateur est peu puissant.

Assurez-vous que le fichier que vous venez de créer existe bel et bien. Quelle taille fait-il ?

- La commande `cp nomfichier nouveaunom` permet de copier le fichier "nomfichier" dans le fichier "nouveaunom".
- La commande `rm nomfichier` permet de supprimer le fichier "nomfichier".
- La commande `mv nouveaunom nomfichier` permet de déplacer le fichier "nouveaunom" vers le fichier "nomfichier". C'est ainsi que l'on peut renommer un fichier en ligne de commande.

Copiez le fichier que vous avez créé. Effacez une des deux copies, et changez le nom de la copie restante. (A chaque étape, assurez-vous que tout se passe comme prévu).

Remontez de manière à vous placer dans le dossier parent du dossier *Essais*. Supprimez ce dossier et son contenu en une seule étape (aidez-vous des pages de manuel de `rmdir` et de `rm`).

2.5 Droits

Les systèmes UNIX sont des systèmes très sécurisés, notamment grâce aux droits et permissions associés à chaque fichier et à chaque dossier. En effet, chaque utilisateur d'un système UNIX possède des droits particuliers, avec des libertés et des restrictions.

Un seul de ces utilisateurs, nommé **root** dispose de tous les droits sur le système, et en est donc l'*administrateur*. Lui seul peut modifier en profondeur le système et a accès à tous les fichiers, même ceux des autres utilisateurs.

Les autres *utilisateurs* (comme vous par exemple) n'ont – par défaut – pas accès aux fichiers des autres utilisateurs, à moins que ceux-ci n'en donnent l'autorisation. En outre, les utilisateurs peuvent être rassemblés en *groupes*, ce qui permet de rendre certains documents visibles à tous les utilisateurs d'un même groupe, et uniquement à eux.

Il existe trois types de droits :
– droits de lecture (r),
– droits d'écriture (w),
– droits d'exécution (x).
Chaque utilisateur a donc des droits spécifiques sur la visualisation du contenu d'un fichier (lecture), sur la modification de ce fichier (écriture), et sur l'exécution de ce dernier. Toutes les combinaisons de droits sont possibles.

Ouvrez un fichier du dossier `/etc/` avec un éditeur de texte. Modifiez-le et enregistrez vos modifications.
Interprétation ?

2.5.1 Visualiser les permissions

On peut voir les permissions accordées à un fichier lorsque l'on tape : `ls -l`. Les 10 premiers symboles correspondent aux autorisations associées à ce fichier : le premier caractère représente les **attributs spéciaux du fichier** (`d` pour "directory" s'il s'agit d'un répertoire, `-` s'il s'agit d'un simple fichier texte, ...). Les trois caractères suivants représentent les permissions dont le **créateur** du fichier dispose sur son fichier (`r` pour lecture, `w` pour écriture et `x` pour exécution). Ensuite, les trois caractères suivants concernent les permissions accordées au **groupe** auquel appartient le fichier. Les trois derniers caractères concernent les permissions accordées au **reste du monde**, c'est-à-dire à l'ensemble des autres utilisateurs.

Tapez la commande `ls -l`.
Listez de même les permissions des fichiers du dossier `/etc/`.
Identifiez les colonnes indiquant le possesseur du fichier, et le groupe du fichier.

2.5.2 Modifier les permissions

On peut modifier les permissions d'un fichier avec la commande :
`chmod u-w nomfichier`
Le premier caractère représente à qui s'applique la modification : `u` pour le créateur du fichier, `g` pour le groupe et `a` pour le reste du monde. Le deuxième représente l'ajout ou la suppression d'une permission (`-` pour la suppression, + pour l'ajout, jusque là tout est logique). Le dernier représente le type de permission à modifier (`r`, `w` ou `x`).
Modifiez différents attributs du fichier que vous avez créé précédemment et observez les résultats.
Supprimez la permission d'exécution d'un répertoire et essayez d'y accéder.
Conclusion ?

2.6 Gestion des processi

2.6.1 Lister les processi

Grossièrement, chaque programme qui s'exécute correspond à un processus (parfois un programme peut lancer plusieurs processi). La commande `ps` permet de prendre connaissance des processi qui tournent actuellement sur l'ordinateur. A chaque processus est associé un PID (*Processus IDentity*).

Affichez tous les processi qui tournent sur votre ordinateur.
Affichez seulement les processi que vous-même avez lancés.
Trouvez la colonne indiquant les PID.

La commande `top` fournit les mêmes informations, avec plus de détails : elle donne ainsi les proportions de la mémoire vive et de la puissance de calcul qui sont utilisées par chacun des processi. Ces informations sont réactualisées en temps réel.

2.6.2 Arrière et premier plans

Lancez un éditeur de texte (pas `vi`) depuis votre console.
Essayez de taper une autre commande dans la console. Que se passe-t-il ?

L'éditeur a été lancé en premier plan, et occupe toutes les ressources de la console : on ne peut donc plus s'en servir. Si on veut à nouveau pouvoir utiliser la console, on a deux solutions :

- soit éteindre l'application normalement – ou violemment en tapant `Ctrl+C` dans la console
- soit mettre l'application en arrière-plan en tapant dans la console `Ctrl+Z` puis une fois que l'invite de commandes est revenue `bg` (*background*).

Avec cette deuxième solution, on dispose à nouveau de la console, et l'application peut continuer à tourner.

Il est possible de lancer une application directement en arrière-plan, en ajoutant le symbole `&` à la fin de la commande.

2.6.3 Procicide

Lorsqu'on n'arrive pas à fermer un processus, on peut le tuer. Pour ce faire, on utilise la commande `kill pid`, qui envoie un signal à l'application ayant le `PID` spécifié.

Lancez un éditeur de texte en arrière plan, puis tuez-le.
Quelle option de `kill` est la plus puissante ?
Par curiosité, relancez un éditeur en arrière plan et tapez dans la console `xkill`. Que fait cette commande ?

3 Approfondissement UNIX

Après avoir vu la création et la gestion de fichiers, dossiers et processus sous UNIX, nous allons voir comment utiliser la puissance d'UNIX pour une gestion plus avancée de votre système.

3.1 Manipuler des fichiers texte

Étant donné que de nombreux fichiers sous UNIX sont de simples fichiers textes, il existe de très nombreuses commandes pour la manipulation simple de ces derniers.

3.1.1 Visualisation de fichiers texte

Si vous ne voulez pas perdre du temps à lancer un éditeur de texte, mais que vous voulez jeter un coup d'oeil à l'intérieur d'un fichier, vous pouvez utiliser la commande `more` (ou `less`), ou la commande `tail` (ou `head`). Ces quatre commandes ne vous permettent toutefois pas de modifier un fichier.

Quelles sont les différences entre `less` et `more` ?

Tapez `less /etc/X11/XF86Config-4` ou bien `less /etc/X11/xorg.conf` si ce dernier n'existe pas.
Naviguez d'avant en arrière dans le fichier.

Quelles sont les différences entre `tail` et `head` ?

Utilisez la commande `tail -20` sur le fichier que vous venez d'inspecter. Que s'est-il passé ?

3.1.2 Concaténation de fichiers

La commande `cat fichier1 fichier2 fichier3` va vous permettre de concaténer ces trois fichiers. Le résultat de cette concaténation sera renvoyé dans la console.

Pour concaténer ces fichiers "colonne par colonne", vous pouvez utiliser la commande `join` ou `paste`

Pour diviser un fichier en morceaux, vous pouvez utiliser la commande `split`

> Par défaut, la plupart des commandes UNIX génère une sortie texte qui est envoyée dans la console pour affichage. Toutefois, il est parfois utile de rediriger cette sortie ailleurs que dans la console.
> Pour rediriger une sortie **vers un fichier**, il suffit de rajouter, à la suite de notre commande, le symbole "**>**" suivi du nom du fichier que l'on veut créer : si ce fichier existe déjà, il sera écrasé. Si l'on veut écrire à la suite de ce fichier, il faudra utiliser le double symbole "**> >**".
> Pour rediriger une sortie **vers une autre commande**, il suffit de rajouter, à la suite de notre première commande, le symbole "**|**" suivi de la deuxième commande.

Que fait la commande `grep` ?

Recherchez le mot *Option* dans le fichier `/etc/X11/XF86Config-4` ou dans `/etc/X11/xorg.conf`, à l'aide de la redirection.

3.2 Automatiser des tâches

3.2.1 Les langages *shell*

Il existe plusieurs langages shell : *bash*, *csh*, *tsh*, *sh*. Les différences entre ces langages sont subtiles et nous ne rentrerons pas dans les détails.

Il faut savoir que, lorsqu'on en connait un, il est très facile d'en apprendre et d'en utiliser un autre. Le shell utilisé par défaut sur les systèmes de l'université est `bash`, et nous ne nous servirons que de celui-ci.

```
#!/bin/bash
```

3.2.2 Petits exercices

On ne peut profiter de la pleine puissance de la ligne de commandes qu'au travers de *scripts shell* qui sont de petits programmes utilisant les fonctions avancées du shell.
Le shell constitue en effet un petit langage de programmation à part entière, avec lequel on peut faire des tests (`if...fi`), des boucles... Nous n'apprendrons pas ici à programmer parfaitement en shell, nous nous contenterons de construire un petit script, comprendre ce qu'il fait, afin de sentir que ces scripts shell peuvent se révéler très utiles au quotidien.

Copiez le script suivant dans un fichier que vous appellerez `note`.

```
fichiermemoire="$HOME/.memoire"

if [ $# -eq 0 ] ; then
echo "Entrez une note, fermez avec ^D: "
cat - >> $fichiermemoire
else
echo "$@" >>$fichiermemoire
fi

exit 0
```

Puis copiez ce second script dans un fichier que vous appellerez `postit`.

```
fichiermemoire="$HOME/.memoire"

if [ $# -eq 0 ] ; then
more $fichiermemoire
else
grep -i "$@" $fichiermemoire | ${PAGER:-more}
fi

exit 0
```

Que font ces scripts ? Exécutez-les.

3.2.3 Alias

Le problème ici est qu'on ne peut lancer ces scripts qu'en spécifiant le chemin pour y accéder. Il serait bien plus pratique de pouvoir les lancer depuis n'importe quel dossier, en utilisant leur nom plutôt que tout le chemin. Pour cela nous avons recours aux `alias`. Les `alias` permettent de fournir un deuxième nom à une commande ; on les utilise typiquement pour donner un nom court et simple à taper à une commande longue dont il est difficile de se rappeler.

Tapez `alias note="$HOME/note"`
Dem même, tapez `alias postit="$HOME/postit"`.
Allez ailleurs dans l'arborescence et tapez `postit`.

On peut enregistrer de manière permanente les alias sur son système. Si cela vous intéresse, demandez à l'encadrant...